

第14回のキーワード

1

アルゴリズム関係

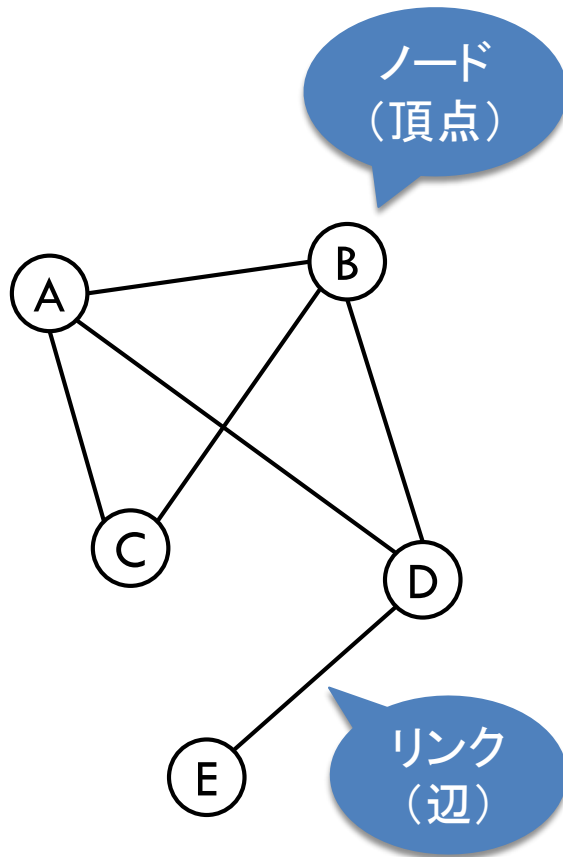
- グラフ構造
(graph structure)
- 無向グラフ
(undirected graph)
- 有向グラフ
(directed graph)
- 重み付きグラフ
(weighted graph)
- 隣接行列/リスト
(adjacency matrix/list)
- ダイクストラ法
(Dijkstra's algorithm)

Java関係

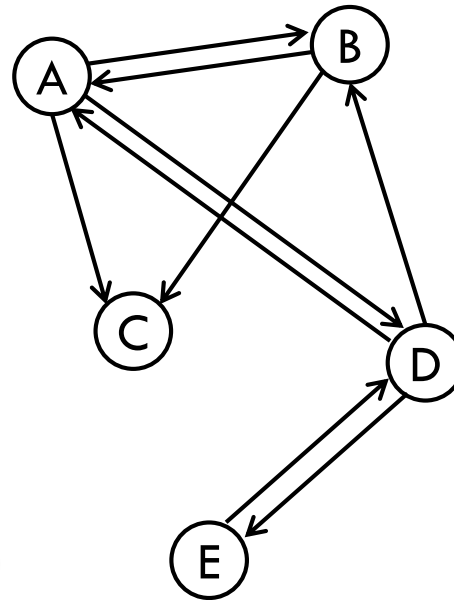
- コレクションフレームワーク
- List, ArrayList, LinkedList
- Set, HashSet, TreeSet
- Map, HashMap, TreeMap
- 匿名クラス
- 関数型インタフェース
- ラムダ式
- forEachメソッド

グラフ構造

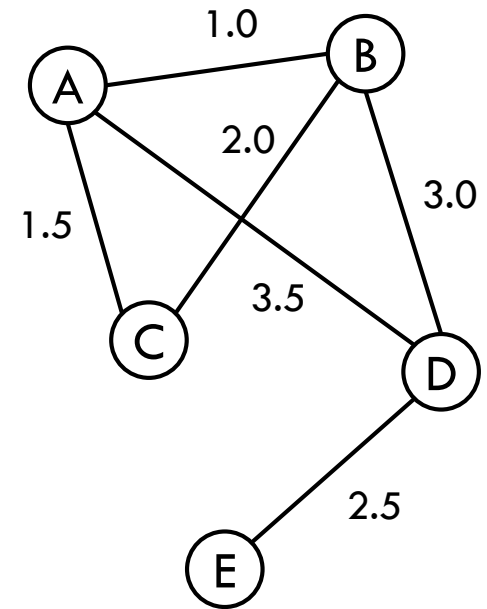
2



無向グラフ



有向グラフ



重み付きグラフ

隣接行列

3

□ ノード間の接続状態を要素とする行列

	A	B	C	D	E
A	0	1	1	1	0
B	1	0	1	1	0
C	1	1	0	0	0
D	1	1	0	0	1
E	0	0	0	1	0

無向グラフ

対称行列になる

$$a_{ij} = a_{ji}$$

	A	B	C	D	E
A	0	1	1	1	0
B	1	0	1	0	0
C	0	0	0	0	0
D	1	1	0	0	1
E	0	0	0	1	0

有向グラフ

一方通行があるので

$$a_{ij} = a_{ji} \text{とは限らない}$$

	A	B	C	D	E
A	∞	1.0	1.5	3.5	∞
B	1.0	∞	2.0	3.0	∞
C	1.5	2.0	∞	∞	∞
D	3.5	3.0	∞	∞	2.5
E	∞	∞	∞	2.5	∞

重み付きグラフ

リンクがない場合は
0または ∞ とする

グラフの探索

4

- 深さ優先探索
 - ▣ 訪問したノードにつながるリンクを優先してたどっていく
 - ▣ 始点から先へ先へと遠くへ探索していく

- 幅優先探索
 - ▣ 先に発見してたどっていないリンクを優先にたどっていく
 - ▣ 始点から探索範囲を段階的に広げていく

- グラフの探索における注意
 - ▣ 同じノードに複数の経路からたどり着く可能性がある
 - ▣ 各ノードが訪問済みなのか記憶しておく必要がある

最短経路問題

5

□ ダイクストラ法

- 重み付きグラフで、ある頂点から全頂点への最短経路を求める
- 理論的に、2点間だけの最短経路を求める確実な方法はない

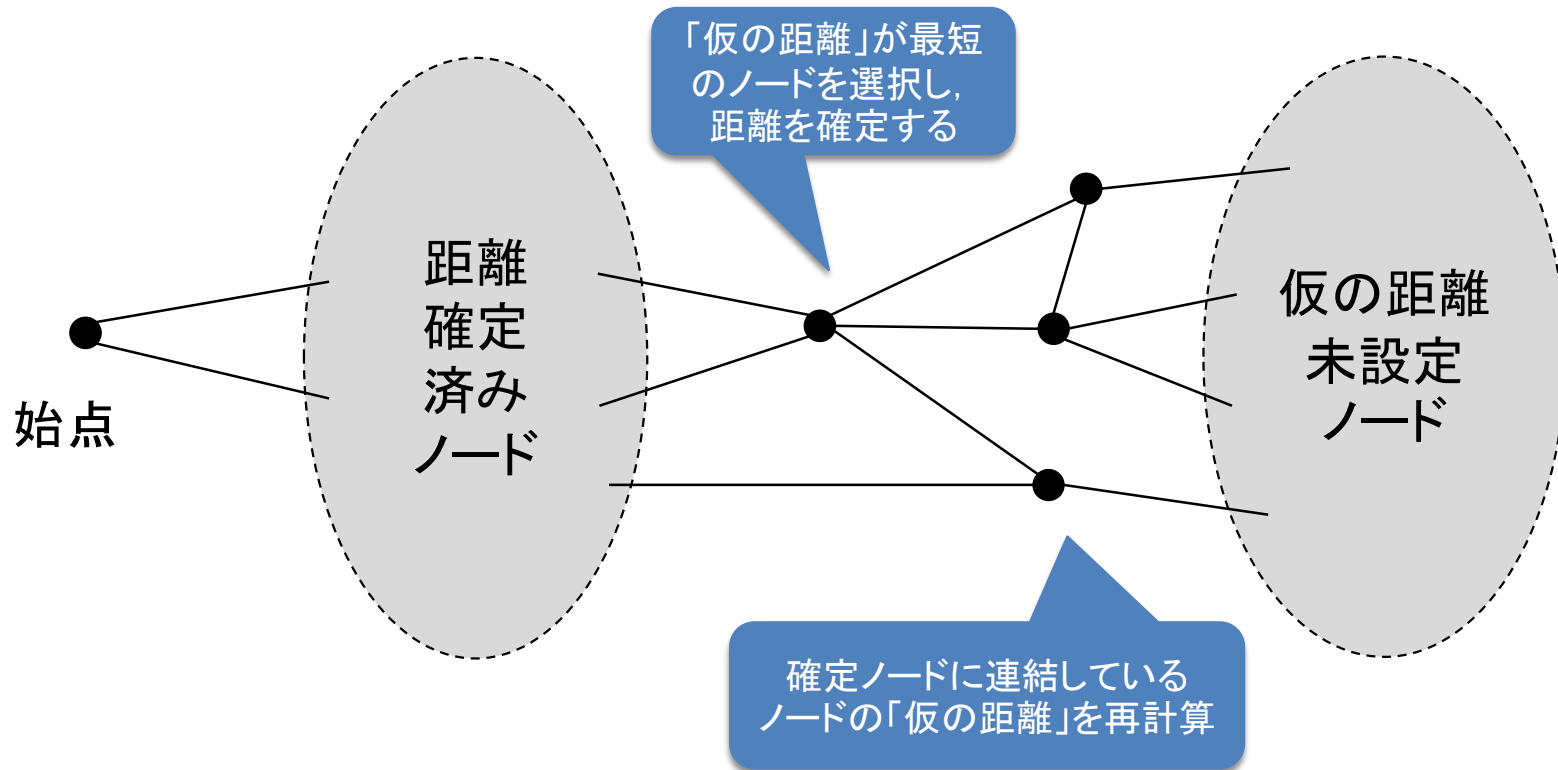
□ アルゴリズム

- 基本方針：未確定の頂点から、1つずつ(始点からの)距離が最短のものを選んで確定済みにし、全頂点が確定したら終わり
 1. 最初は、全頂点の(始点からの)距離を未確定とする
 2. 全頂点が確定するまで以下を繰り返す
 - ① 未確定の頂点の中から、始点からの「仮の距離と経路」が最短のもの v を選択する(最初は始点を選択される)
 - ② v の距離と経路は確定済みとする
 - ③ v に直接連結している頂点の「仮の距離と経路」を更新する

ダイクストラ法

6

- 「仮の距離」が最短のノードを選んで確定していく
 - ▣ 重みなしグラフの場合，幅優先探索と同様になる



Javaコレクションクラス

7

クラス	内部構造	親インタフェース	インタフェースの説明と代表的なメソッド
<code>ArrayList<E></code>	配列	<code>List<E></code>	データを一行に並べて格納する(第6回, 第10回で説明)
<code>LinkedList<E></code>	連結リストの一種		
<code>HashSet<E></code>	ハッシュテーブル	<code>Set<E></code>	要素が重複しないようにデータを格納する(集合) <code>add(data)</code> , <code>contains(data)</code> , <code>remove(data)</code> , <code>size()</code> , <code>isEmpty()</code>
<code>TreeSet<E></code>	2分探索木の一種		
<code>HashMap<K, V></code>	ハッシュテーブル	<code>Map<K, V></code>	キーと値のペアでデータを格納する(写像, 辞書, 連想配列) <code>put(key, value)</code> , <code>get(key)</code> , <code>remove(key)</code> , <code>isEmpty()</code> , <code>size()</code> , <code>containsKey(key)</code> , <code>entrySet()</code> , <code>keySet()</code> , <code>values()</code>
<code>TreeMap<K, V></code>	2分探索木の一種		
<code>Map.Entry<K, V></code>			<code>Map<K, V></code> の各要素を表す内部クラス <code>getKey()</code> , <code>getValue()</code>

親インタフェースがデータ構造に対する抽象的な操作方法を提供する
なるべく親インタフェースを介して用いると具体的な実装を変更可能になる

匿名クラスとラムダ式

8

□ 匿名クラス

- 抽象クラスまたはインタフェースを継承した“使い捨て”の子クラスとそのインスタンスを定義する方法
- class文を使わないのでクラスに名前がつかない
- 形式: `new 親クラス名 { クラスメンバの定義 }`

または, 親インタフェース名

□ ラムダ式

- 一般的には, 式として関数を定義する記法
- Javaの場合, メソッドが1つしかないインタフェースを実装した匿名クラスのインスタンスを生成する
- 形式: `(引数並び) -> { 唯一のメソッドの処理 }`